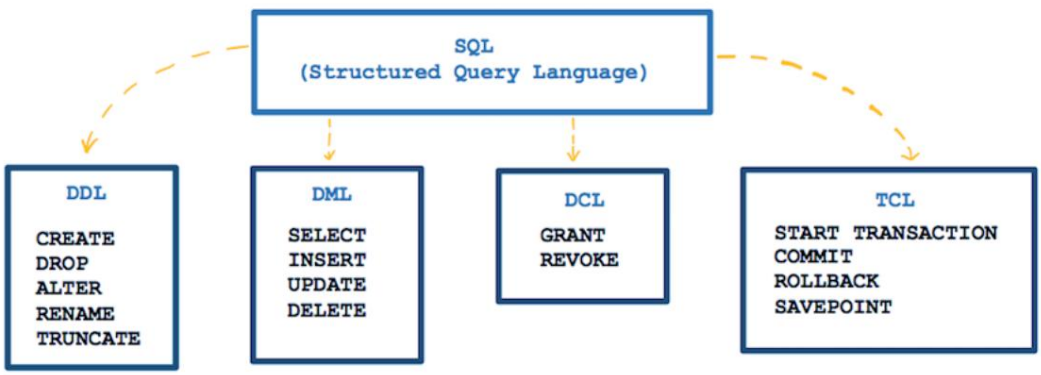
Contenido



Introducción. Procedimientos almacenados y funciones

Los procedimientos y funciones almacenados son funcionalidades a partir de la versión de MySQL 5.0.

> Un procedimiento o función almacenado es un conjunto de comandos SQL que pueden almacenarse en el servidor; se almacenan asociados a una base de datos. Una vez que se hace, los clientes no necesitan relanzar los comandos individuales; pueden en su sitio referirse al procedimiento almacenado.

> A una función o procedimiento también se le llama rutinas.
> Son programas que pueden invocarse por parte de un cliente de la BD autorizado.

Algunas situaciones en las que pueden ser particularmente útiles:

• Cuando múltiples aplicaciones cliente se escriben en distintos lenguajes o funcionan en diferentes plataformas, pero necesitan realizar la misma operación en la base de datos.

• Cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos o funciones para todas las operaciones comunes.

Esto proporciona un entorno seguro y consistente, y los procedimientos pueden asegurar que cada operación se loguea apropiadamente.

En este entorno, las aplicaciones y los usuarios no obtendrían ningún acceso directo a las tablas de la base de datos, sólo pueden ejecutar algunos procedimientos o funciones almacenadas.

Estas aplicaciones o rutinas almacenadas pueden mejorar el rendimiento puesto que se necesita enviar menos información entre el servidor y el cliente. El intercambio que hay es que aumenta la carga del servidor de la base de datos ya que la mayoría del trabajo se realiza en la parte del servidor y no en el cliente.

Los procedimientos almacenados le permiten tener bibliotecas o funciones en el servidor de base de datos. Esta característica es compartida por los lenguajes de programación modernos que permiten este diseño interno, por ejemplo, usando clases. Usando estas características del lenguaje de programación cliente es beneficioso para el programador incluso fuera del entorno de la base de datos.

MySQL sigue la sintaxis SQL:2003 para procedimientos almacenados, que también usa IBM DB2.

Introducción al lenguaje de programación

Sentencia SIETE. Variables de usuario

```
SET var_name = expr [, var_name = expr] ...
```

SET es una sentencia que permite asignar un valor a una variable.

Una variable de usuario permite almacenar un valor y referirnos a él después. También pueden pasarse de una sentencia a otra. Las variables de usuario no pueden ser vistas por otros usuarios y desaparecen cuando la conexión se cierra.

Las variables de usuario comienzan siempre por el carácter @. Para crear una variable de usuario la hay basta con inicializarla:

Podemos utilizar como operador de asignación tanto el signo = como el signo := cuando las definimos con SIETE

Ejemplos de uso:.

```
SIETE @VarGama="Herramientas";

SELECT *
FROM Productos
WHERE Gama=@VarGama;

SET @numeroProd = SELECT COUNT(*)
                  FROM Productos;
```

Para utilizar una variable de usuario:.

- Podemos utilizar las variables de usuario en cualquier lugar donde se puedan utilizar expresiones
- No asignar un valor a una variable de usuario en una parte de una sentencia y usarla en una otra parte de la misma sentencia. Por ejemplo: no debe asignarse un valor a una variable en SELECT y hacer referencia a ella en HAVING, GROUP BY o ORDER BY. Puede dar resultados inesperados.
- Si hacemos referencia a una variable sin inicializar con ningún valor, su valor es NULL y de tipos cadena.

Cómo visualizar el valor de una variable

La sentencia SELECT la podemos utilizar para visualizar el valor de una o varias variables. La sintaxis es la siguiente:

```
SELECT var1 [, var2...]
```

Así, en el ejemplo anterior, hemos creado las variables de usuario @VarGama y @numeroProd. Ahora podemos ver su valor:

```
SELECT @VarGama, @numeroProd;
```

Sentencia SELECT ... INTO

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

Esta sintaxis SELECT almacena los valores columnas seleccionadas directamente en variables. Por tanto, la consulta SÓLO PUEDE DEVOLVER UNA FILA con el mismo número de columnas que de variables.

Por ejemplo:

```
SELECT COUNT(*), AVG(PrecioVenta), MAX(PrecioVenta) INTO @a, @b, @c
FROM Productos;

SELECT @a, @b, @c;
```

Sentencias para rutinas: procedimientos o funciones almacenadas

DELIMITER

"

El delimitador en MySQL (lo que separa una sentencia de otra) es por defecto el " ;

Una rutina suele tener más de una sentencia. Por lo que para poder escribir una rutina que incluya varias sentencias debemos asignar la función de delimitador entre sentencias a otro carácter (o conjunto de ellos sin espacios en blanco), como por ejemplo la barra (|). Una vez cambiado, utilizaremos el punto y cómo (;) para separar las sentencias dentro de la rutina y el nuevo delimitador para indicar que hemos terminado la rutina o sentencia.

Podemos cambiar el delimitador cuando lo consideremos. Nunca dentro de una rutina.

En este ejemplo, estamos configurando los caracteres $$ como los separadores entre las sentencias SQL.

```
DELIMITER $$
```

Después de cambiarlo, deberemos usar $$ para separar las sentencias y el ; sólo servirá para separar sentencias que están dentro de una rutina.

En este ejemplo volvemos a configurar que el carácter separador es el punto y coma.

```
DELIMITER;
```

Sentencia compuesta BEGIN ... END

```
BEGIN
 [lista_sentencias]
END
```

La sintaxis BEGIN ... END se utiliza para escribir sentencias que pueden aparecer en el interior de rutinas almacenadas. Una sentencia compuesta puede contener múltiples sentencias, cerradas por las palabras BEGIN y END. Por ejemplo:

```
DELIMITER $$
CREATE PROCEDURE ejemplo1()
BEGIN
        SET @a= 'Esto es una prueba';
        SIETE @b=6*8;
        SELECT @b, @a;
END $$
CALL ejemplo1()$$
```

Declarar variables locales con DECLARE

```
DECLARE var_name[,...] type [DEFAULT value]
```

Este comando se usa para declarar variables locales. Para proporcionar un valor predeterminado para la variable, incluye una cláusula DEFAULT

El valor puede especificarse como expresión, no necesita ser un literal. Si la cláusula DEFAULT no está presente, el valor inicial es NULL.

La visibilidad de una variable local está dentro del bloque BEGIN…END donde está declarado.

Si hay blog DECLARE, debe ir inmediatamente después de BEGIN.

PROCEDIMIENTOS

Se trata de un conjunto de instrucciones SQL que se ejecutan cuando el procedimiento es nombrado.

Los procedimientos son rutinas o subprogramas compuestos por un conjunto de sentencias, agrupadas lógicamente para realizar una tarea específica, que se guardan en la base de datos y que se ejecutan como una unidad cuando el procedimiento es invocado.

Creación de procedimientos.

Para crear procedimientos se utiliza la siguiente sintaxis:

```
CREATE PROCEDURE NombreProcedimiento ([parámetro1 [, parámetro2,..]])
CuerpoDelProcedimiento
```

Si el CosDelProcedimiento agrupa varias sentencias, comienza con BEGIN y termina con la sentencia END (cuando hay más de una sentencia) y consta de varias instrucciones. Cada una termina con punto y coma (;).

Si en el Cuerpo del Procedimiento deben declararse variables locales (sentencia DECLARE), se hace inmediatamente después de BEGIN.

Sentencia CALL

```
CALL sp_name([parameter[,...]])
```

El comando CALL invoca un procedimiento definido previamente con CREATE PROCEDURE.

Parámetros de entrada, salida y entrada/salida

En los procedimientos almacenados podemos tener tres tipos de parámetros:

- Entrada: Se indican poniendo la palabra reservada IN delante del nombre del parámetro o no poniendo nada: es la opción por defecto. De estos parámetros el procedimiento sólo toma el valor, es decir, cuando el procedimiento finalice estos parámetros tendrán el mismo valor que tenían cuando se hizo la llamada al procedimiento. En programación sería equivalente al paso por valor de un parámetro. Cuando el procedimiento se invoca un parámetro IN puede ser un literal o variable.
- Salida: Se indican poniendo la palabra reservada OUT delante del nombre del parámetro. Estos parámetros cambian su valor dentro del procedimiento. Cuando se hace la llamada al procedimiento comienzan con un valor inicial y cuando finaliza la ejecución del procedimiento pueden acabar con otro valor distinto. En programación sería equivalente al paso por referencia de un parámetro. Cuando el procedimiento se invoca un parámetro OUT solo puede ser una variable.
- Entrada/Salida: Es una combinación de los tipos IN y OUT. Estos parámetros se indican poniendo la palabra reservada INOUT delante del nombre del parámetro. . En programación sería equivalente al paso por referencia de un parámetro. Cuando el procedimiento se invoca un parámetro IN/OUT sólo puede ser una variable.

A efectos prácticos, nos da igual poner OUT que INOUT

Ejemplo en la BD Jardinería:

```
-- En este ejemplo no hace falta BEGIN-END porque sólo tiene una instrucción
--Tampoco cambiar el DELIMITER

CREATE PROCEDURE contar_productos(IN PGama VARCHAR(50), OUT total INT
UNSIGNEDO)

        SELECT COUNT(*) INTO total
        FROM productos
        WHERE gama = PGama;


SIETE @cuenta=1;
CALL contar_productos('Herramientas', @cuenta);
SELECT @cuenta;
```

```
-- También podríamos invocarlo con dos variables como parámetros
SIETE @varGama='Aromáticas';
CALL contar_productos(@varGama, @cuenta);
SELECT @cuenta;
```

```
--Otro ejemplo de paso de parámetros IN o OUT

DELIMITER $$
CREATE PROCEDURE EjemploParam(a int, OUT b int)
BEGIN
        SIETE a=100;
        SIETE b=100;
END $$

SET @primer=1, @segundo=1 $$
CALL EjemploParam(@primero, @segundo) $$
SELECT @primero, @segundo $$
```

EJERCICIOS PROCEDIMIENTOS

Procedimientos con sentencias SQL (Jardinería)

1. Escribe un procedimiento que reciba el nombre de un país como parámetro de entrada y realice una consulta sobre la mesa CLIENTES para mostrar todos los clientes que existen en la mesa de ese país (sentencia SELECT) .

2. Escribe un procedimiento que reciba como parámetro de entrada una forma de pago, que será una cadena de caracteres (Ejemplo: PayPal, Transferencia, etc). Y retorno como salida el pago de máximo valor realizado para esa forma de pago. Deberá hacer uso de la mesa PAGOS.

3. Escribe un procedimiento que reciba como parámetro de entrada una forma de pago, que será una cadena de caracteres (Ejemplo: PayPal, Transferencia, etc). Y retorno como salida los siguientes valores teniendo en cuenta la forma de pago seleccionada como parámetro de entrada:

• el pago de máximo valor,

• el pago de mínimo valor,

• el valor medio de los pagos realizados,

• la suma de todos los pagos,

• el número de pagos realizados para esa forma de pago.

4. Escribe un procedimiento que se llame calcular_max_min_media, que reciba como parámetro de entrada el nombre de la gama de un producto (tabla PRODUCTOS) y retorne como salida tres parámetros. El precio máximo, el precio mínimo y la media de los PrecioVenta de los productos que existen en esa gama.

5. Procedimiento que dado un número entero (entre 0 y 100) y un CódigogoCliente pasados como a parámetros incremente en el porcentaje correspondiente el campo LimiteCredito de la fila correspondiente de la mesa Clientes.
6. Procedimiento que dado un CodigoPedido pasado como parámetro elimine el pedido de la tabla Pedidos. Para ello deberá eliminar primero las filas relacionadas en DetallePedidos.

Nota: para ampliar, puede consultar el manual o

https://dev.mysql.com/doc/refman/5.7/en/faqs-stored-procs.html